H6－214690

Purpose: To enable synchronization control of a multimedia object in detail.

Constitution: A state 1 in a basic class A is constituted of states of state numbers 1.0, 1.1, and 1.2. Among them, a state number (n) of the basic class A is selected, and further subdivided into a plurality of classes. The subdivided classes are located in a derived class B of the basic class A. For instance, the derived class B is constituted of the states of the state numbers (n.0), (n.1), and (n.m). Similarly, regarding one of the state numbers, for example, the state number (n.m), a derived class C of the derived class B is indicated by a set of states including state numbers (n.m.0), (n.m.1), and (n.m.k). In such a manner, the classes are hierarchically defined.

WN-2449

(19)日本国特許庁（ＪＰ）　(12)**公開特許公報**（Ａ）　(11)特許出願公開番号

特開平6－214690

(43)公開日　平成６年(1994)８月５日

(51)Int.Cl.[5]		識別記号	庁内整理番号	ＦＩ	技術表示箇所
Ｇ０６Ｆ	3/00		Ｚ　7165－5Ｂ		
	9/44	３３０	Ｚ　9193－5Ｂ		

審査請求　未請求　請求項の数５　ＦＤ　（全 14 頁）

(21)出願番号　特願平5－21728

(22)出願日　平成５年(1993)１月14日

(71)出願人　000002185
ソニー株式会社
東京都品川区北品川６丁目７番35号

(72)発明者　山岸　靖明
東京都品川区北品川６丁目７番35号　ソニー株式会社内

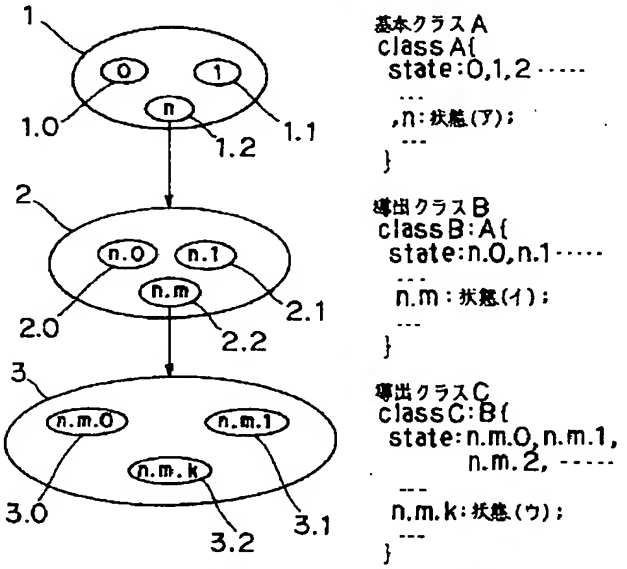
(74)代理人　弁理士　稲本　義雄

(54)【発明の名称】　情報同期制御方法、情報伝送装置および情報出力装置

(57)【要約】

【目的】　マルチメディアオブジェクトのより細かな同期制御を可能とする。

【構成】　基本クラスＡにおける状態１は、状態番号１．０，１．１および１．２の状態により構成される。このうち、例えば基本クラスＡの導出クラスＢにおける基本クラスＡの状態番号ｎの状態を細分化した状態の集合は、状態番号ｎ．０，ｎ．１，ｎ．ｍの状態により構成される。同様に、導出クラスＢの導出クラスＣは、導出クラスＢの状態番号ｎ．ｍの状態を細分化した状態の集合３で表され、この集合３は、状態番号ｎ．ｍ．０，ｎ．ｍ．１，ｎ．ｍ．２およびｍ．ｍ．ｋの状態により構成される。このように、クラスが階層的に定義される。



クラスの階層的定義による状態遷移の継承

1

【特許請求の範囲】

【請求項1】　複数のメディアからの情報を出力する際の各情報間の同期関係を制御する情報同期制御方法において、

各情報のメディアからの出力状態と、前記出力状態の遷移を階層的に定義し、前記出力状態とその遷移を利用したスクリプトをもとに、各情報の出力状態の同期を制御することを特徴とする情報同期制御方法。

【請求項2】　前記各情報のメディアからの出力状態と、その遷移は、基本クラス、前記基本クラスの下位の導出クラス、または他の導出クラスの下位の導出クラスにより定義されることを特徴とする請求項1に記載の情報同期制御方法。

【請求項3】　前記基本クラスと導出クラスは、前記メディアの状態を定義するステートと、前記状態の遷移を表わすイベントにより定義されることを特徴とする請求項2に記載の情報同期制御方法。

【請求項4】　情報のメディアからの出力状態とその遷移を階層的に定義するとともに、前記出力状態とその遷移を利用したスクリプトを作成する第1の作成手段と、前記メディアのオブジェクトデータを作成する第2の作成手段と、

前記第1の作成手段と第2の作成手段の出力を合成して伝送する伝送手段とを備えることを特徴とする情報伝送装置。

【請求項5】　入力信号から、階層的に定義された情報のメディアからの出力状態とその遷移を利用したスクリプトと、メディアのオブジェクトデータとを分離する分離手段と、

前記分離手段により分離されたスクリプトを解釈する解釈手段と、

前記解釈手段の出力に対応して、前記分離手段により分離された前記オブジェクトデータを処理する処理手段とを備えることを特徴とする情報出力装置。

【発明の詳細な説明】

【0001】

【産業上の利用分野】本発明は、マルチメディア情報を表示／再生する情報処理装置において、情報要素間の同期関係を制御する情報同期制御方法、情報伝送装置および情報出力装置に関する。

【0002】

【従来の技術】ワークステーション、記憶装置、通信装置などの発展に伴い、テキストデータや静止画像ばかりでなく、サウンドシークエンス、オーディオビジュアル等の新しい表現メディア（相互交換されるデータの種類、符号化された形式で記述される情報そのもの）を利用したアプリケーションプログラムが普及しつつある。

【0003】この表現メディアの多様化により、各々のメディアを組み合わせた使い方、すなわち、マルチメディア（複数種類の表現メディア）を扱う要求が増加し、

2

それとともに、マルチメディア情報を異なるプラットフォーム間で共有するための標準的な情報の交換形式の規定が望まれている。

【0004】その1つの例が、情報要素（単一の表現メディアによって表される塊、以下、メディアオブジェクトとも記す）間の表示／再生における同期制御方式（時間依存関係の制御方式）の規定である。

【0005】マルチメディア同期制御の制御方式には、大きく分けて、ビット／パケットレベル、オブジェクト（1つの意味のまとまり単位）レベルの2つの段階が考えられる。前者の例にはＣＣＩＴＴ勧告Ｈ．２２１等があり、後者の例にはＩＳＯ／ＩＥＣ／ＤＩＳ１０７４４（ＨｙＴｉｍｅ）が挙げられる。

【0006】ＨｙＴｉｍｅ等、従来のオブジェクトレベルの同期制御方式においては、個々のメディアオブジェクトの出現位置／領域が、空間および時間軸上にアドレッシングされている。つまり、表示／再生位置が静的にスケジュールされている。

【0007】

【発明が解決しようとする課題】マルチメディア情報を表示／再生する際の情報要素間の同期関係を制御する方式のうち、オブジェクトレベルの同期制御方式においては、表示／再生位置を静的にスケジュールする方法が一般的である。

【0008】しかしながら、このような静的なスケジューリングのみをサポートする制御方式は、ユーザからの対話のタイミングによる同期を制御する能力が貧弱である課題を有している。

【0009】例えば、いま、ここに、マルチメディア情報表示／再生装置があり、この装置は同期関係を記述したスクリプト（情報要素間の表示／再生制御のシナリオを記述したプログラム）を読み込み、それを実行することが可能であるものとする。そして、そのスクリプトには、ディスプレイ装置の画面上にボタンと動画表示領域を配置し、ユーザが画面上のボタンを押すと、動画の再生が行なわれるというように記述されているものとする。

【0010】ただし、そのスクリプトは、静的なスケジューリングのみをサポートする同期制御記述方式で記述されている。ここでいう、静的にスケジューリングするとは、”ボタンが押された場合には、動画をそのデータの最初から再生する。”というような記述の仕方のことをいう。

【0011】従って、この場合、動画の再生途中にボタンを押したとき、動画が再生途中であるか否かにかかわらず、動画の最初から再生が行なわれ、再生をそのまま続けるか、最初から再生し直すかの選択ができない。つまり、ボタンを押したときの動画の表示状態により、再生の仕方を細かく制御することができないという問題点がある。

【００１２】従って、ユーザとの対話管理を含めた、マルチメディアオブジェクトの同期制御を効果的に記述するためには、スクリプトに、現在表示中のメディアオブジェクトの表示の状態に依存した同期制御を記述する能力（動的なスケジューリングの能力）が求められる。

【００１３】さらに、メディアが異なると、その状態の種類が異なるため、すべてのメディアに共通な状態というものを定義することができるとともに、個々のメディアの特性に依存する状態を定義することができる能力を有することが望ましい。

【００１４】例えば、静止画と動画を例に取ると、静止画は時間に依存しないデータ（離散メディア）であるため、それがもつ表示の状態は大きく分けると、”表示中”、または”表示されていない”、という２つの状態のいずれかである。これに対して、動画の場合は、それが時間に依存するデータ（連続メディア）であるため、”表示中”、または”表示されていない”、という２つの状態に区分される他、”表示中”はさらに、”通常の速度での再生”と、”倍速再生”に区分される。

【００１５】本発明は、以上のような問題点に鑑みてなされたものであり、マルチメディア情報の同期制御を行なうに際し、マルチメディアオブジェクトの表示中のマルチメディア情報要素の状態に依存した細かな同期制御を可能とすることを目的としている。

【００１６】

【課題を解決するための手段】請求項１に記載の情報同期制御方法は、複数のメディアからの情報を出力する際の各情報間の同期関係を制御する情報同期制御方法において、各情報のメディアからの出力状態と、出力状態の遷移を階層的に定義し、出力状態とその遷移を利用したスクリプトをもとに、各情報の出力状態の同期を制御することを特徴とする。

【００１７】各情報のメディアからの出力状態と、その遷移は、基本クラス、基本クラスの下位の導出クラス、または他の導出クラスの下位の導出クラスにより定義することができる。

【００１８】また、基本クラスと導出クラスは、メディアの状態を定義するステートと、状態の遷移を表わすイベントにより定義することができる。

【００１９】請求項４に記載の情報伝送装置は、情報のメディアからの出力状態とその遷移を階層的に定義するとともに、出力状態とその遷移を利用したスクリプトを作成する第１の作成手段としてのクラス定義スクリプト作成部１１と、メディアのオブジェクトデータを作成する第２の作成手段としてのメディアオブジェクトデータ作成部１２と、クラス定義スクリプト作成部１１とメディアオブジェクトデータ作成部１２の出力を合成して伝送する伝送手段としての転送制御部１３とを備えることを特徴とする。

【００２０】請求項５に記載の情報出力装置は、入力信号から、階層的に定義された情報のメディアからの出力状態とその遷移を利用したスクリプトと、メディアのオブジェクトデータとを分離する分離手段としての転送制御部１６と、転送制御部１６により分離されたスクリプトを解釈する解釈手段としてのスクリプト解釈実行部１７と、スクリプト解釈実行部１７の出力に対応して、転送制御部１６により分離されたオブジェクトデータを処理する処理手段としての表示制御部１８とを備えることを特徴とする。

【００２１】

【作用】本発明の情報同期制御方法、情報伝送装置および情報出力装置においては、各情報のメディアからの出力状態と、出力状態の遷移を階層的に定義し、出力状態とその遷移を利用したスクリプトをもとに、各情報の出力状態の同期が制御される。従って、マルチメディアオブジェクトのより細かな同期制御が可能となる。

【００２２】

【実施例】以下に実施例について説明する。本実施例においては、メディアがクラスとして定義され、クラスにおいて、メディアの状態遷移がイベントとして定義される。クラスは、さらに基本クラスと導出クラスとに区分される。

【００２３】

基本クラスのシンタクスは、以下のように定義される。

```
class <クラス名(文字列)> {
  state:
    [<状態番号(正の整数)> : <状態名(文字列)>;]*
  event:
    [<イベント番号(正の整数)> : <イベント名(文字列)> :
     <状態番号(正の整数)> → <状態番号(正の整数)>;]*
}
```

【００２４】”ｓｔａｔｅ”の項には、状態番号と対応する状態名の組を記述する。状態番号は、この”ｓｔａｔｅ”の項内で一意な整数である。”ｅｖｅｎｔ”の項には、”＜状態番号（正の整数）＞→＜状態番号（正の整数）＞”で示される＜状態番号（正の整数）＞→＜状態番号（正の整数）＞の状態遷移を引き起こすイベントの名前とイベント番号が組にして列挙される。イベント番号は、この”ｅｖｅｎｔ”の項内で一意な整数である。なお、”［．．．］＊”は、．．．が１つ以上出現することを表す。なお、後述するように、状態番号”

０”として設定された状態は、初期状態とされる。

【００２５】導出クラスのシンタクスは、以下のように定義される。

```
class <クラス名(文字列)> : <基本クラス名または導出クラス名(文字列)>
{
  state:
   [<状態番号(正の整数列)> : <状態名(文字列)>;]*
  event:
   [<イベント番号(正の整数)> : <イベント名(文字列)> :
    <状態番号(正の整数列)> → <状態番号(正の整数列)>;]*
}
```

【００２６】この”クラス名”で識別される導出クラスは、親クラスである”基本クラス名”または”導出クラス名”で識別されるクラスの状態遷移と、それを引き起こすイベントを継承する。この場合、”状態番号”のフォーマットは、”．”で区切られた２つ以上の正の整数列で表わされ、最も右側の整数の左に隣接する整数は、１つ上位の親クラスの状態番号を表わす（後述の例を参照のこと）。すなわち、導出クラスの記述は、親のクラス（基本クラスまたは導出クラス）のある状態を、さらに細分化した状態を定義することになる。

【００２７】例えば、ある基本クラスにおいて、状態番号ｎ（ｎは正の整数）が定義される場合、その導出クラス（子クラス）において、親クラスの状態番号ｎに対応する状態をさらに細分化して定義するときは、状態番号をｎ．ｍ（ｎ、ｍは正の整数）とする。

【００２８】さらに、その導出クラス（子クラス）の導出クラス（孫クラス）において、子クラスの状態番号ｎ．ｍに対応する状態をさらに細分化して定義する場合は、状態番号をｎ．ｍ．ｋ（ｎ、ｍ、ｋは正の整数）とする。

【００２９】以上のことを、図１を参照してさらに説明する。図１において、１は、基本クラスＡにおける状態の集合を表し、１．０は状態番号０の状態を表し、１．１は状態番号１の状態を表し、１．２は状態番号ｎの状態を表している。また、２は、基本クラスＡの導出クラスＢにおける、Ａの状態番号ｎの状態１．２を細分化した状態の集合を表し、２．０は状態番号ｎ．０の状態を表し、２．１は状態番号ｎ．１の状態を表し、２．２は状態番号ｎ．ｍの状態を表している。同様に、３は、導出クラスＢの導出クラスＣにおける、Ｂの状態番号ｎ．ｍの状態２．２を細分化した状態の集合を表し、３．０は状態番号ｎ．ｍ．０の状態を表し、３．１は状態番号ｎ．ｍ．１の状態を表し、３．２は状態番号ｎ．ｍ．ｋの状態を表している。

【００３０】上述のクラス定義に記述された状態遷移に従うオブジェクトの宣言と、それらの表示制御手続きがスクリプトに記述される。スクリプトのシンタクスは以下のようになされる。

```
script〈スクリプト名〉{
 〈オブジェクト定義部〉
 〈イニシャルメッセージパッシング記述部〉
 〈イベントメッセージパッシング記述部〉
 }
```

【００３１】オブジェクト定義部は、次のように規定される。

```
<オブジェクト定義部> =
  [<オブジェクト変数> = new <クラス名>(<内容部識別子>);]*
```

【００３２】オブジェクト定義部は、このスクリプトで記述の対象となるオブジェクトを定義する部分である。オブジェクト変数は、指定されたクラスのインスタンスを特定する識別子となる。

【００３３】内容識別子は、次式で表わされるように、そのオブジェクトの表現メディアそのもののデータを特定する識別子である。

```
<内容識別子> =
content_identifier = <正の整数>
```

【００３４】イニシャルメッセージパッシング記述部は、次式で表わされ、そのスクリプトが起動されたとき、一番最初に実行される部分である。

```
<イニシャルメッセージパッシング記述部> =
```

```
<イベントメッセージパッシング記述部> =
 [switch (<イベント判定部>) {
```

<メッセージパッシング記述部>

【００３５】メッセージパッシング記述部は次式で表わされる。

```
<メッセージパッシング記述部> =
do {[<イベント名> → <オブジェクト変数>;]*}
```

【００３６】この式により、オブジェクト変数で特定されるオブジェクトに対してイベント名で特定されるイベントが送られる。その結果、対象オブジェクトの状態遷移が起こる。このイベント名は、対象のオブジェクトのクラス定義において定義されていなければならない。

【００３７】イベントメッセージパッシング記述部は次式で表わされる。

[case（<状態判定部>）<メッセージパッシング記述部>]*
}]*

【００３８】イベント転送の検出（後述）が起こったとき、すべてのｓｗｉｔｃｈ文のイベント判定部に指定されている条件のテストが行なわれ、それが成立した場合、ｃａｓｅ文の状態判定部に指定されている条件のテストが行なわれる。テストした条件が成立した場合、メッセージパッシング記述部に記述されているメッセージパッシングが行なわれる。

【００３９】ｃａｓｅ文の状態判定部はなくてもよく、その場合には、ただちにメッセージパッシングが行なわれる。

【００４０】イベント判定部は次式で表わされる。

<イベント判定部> ＝
<オブジェクト変数>．<イベント判定メソッド>

【００４１】オブジェクト変数で特定される対象オブジェクトにおいて当該イベントが起こったか否かをテストする。起こっていれば、このイベント判定部が成立することとなる。

【００４２】イベント判定メソッドは次式で表わされる。

<イベント判定メソッド> ＝
isSelected()、または、getMessage(<イベント名>)

【００４３】ｉｓＳｅｌｅｃｔｅｄ（）メソッドは、対象オブジェクトに対して”ユーザの選択”（後述）が起こったか否かをテストするものであり、ｇｅｔＭｅｓｓａｇｅ（）メソッドは、引き数<イベント名>で特定されるイベントが対象オブジェクトにおいて起こったか否かをテストするものである。

【００４４】状態判定部は次式で表わされる。

<状態判定部> ＝
<オブジェクト変数>．<状態判定メソッド>
[<論理演算子><オブジェクト変数>．<状態判定メソッド>]*

【００４５】これは、オブジェクト変数で特定される対象オブジェクトにおける状態をテストするものであり、対象オブジェクトが下記の状態判定メソッドの引き数<状態番号>で特定される状態にあるか否かをテストする。

<状態判定メソッド> ＝
testState(<状態番号>)

【００４６】なお、＜論理演算子＞　はＣ言語の論理演算子（＆＆（および），｜｜（または））と同じ意味である。

【００４７】上述のクラス定義の表記構文と、スクリプトの表記構文にもとづいて、具体的なクラス定義とスクリプトの例を送り側で記述し、それらを伝送媒体を介して受け側のマルチメディア情報表示／再生情報処理装置に伝送し、受け側において、送り側で意図したシナリオにもとづいて、マルチメディア情報を同期制御する方法を以下に説明する。

【００４８】最初に、メディア（Ｍｅｄｉａ）クラスを定義する。このクラスに属するオブジェクトのとりえる状態として、例えば、次の２つの状態を定義する。

あ）表示待機中（ＷＡＩＴ）
い）表示中（ＲＵＮ）

【００４９】このオブジェクトが生成されたときの初期状態は”表示待機中（ＷＡＩＴ）”とする。

【００５０】また、イベントとして、次の２つを定義する。

ア）ｒｕｎ
イ）ｓｔｏｐ

【００５１】以上のメディアの定義をまとめると、次のようになる。

```
class Media {
  state:
    0: WAIT;
    1: RUN;
  event:
    0:run: 0→1;
    1:stop: 1→0;
}
```

【００５２】図２に示すように、このオブジェクトが状態”ＷＡＩＴ”の時に、”ｒｕｎ”というメッセージを受けると、このオブジェクトの状態が、”ＲＵＮ”に遷移する。また、オブジェクトが状態”ＲＵＮ”の時に、”ｓｔｏｐ”というメッセージを受けると、このオブジェクトの状態が”ＷＡＩＴ”に遷移する。この状態遷移を表にまとめると、表１に示すようになる。

【００５３】

【表１】

〈Ｍｅｄｉａクラスの状態遷移表〉

	イベント	
状態	ｒｕｎ	ｓｔｏｐ
ＷＡＩＴ	ＲＵＮ	－
ＲＵＮ	－	ＷＡＩＴ

【００５４】次に、上述のメディアクラスから導出される連続メディア（Ｃｏｎｔｉｎｕｏｕｓ　Ｍｅｄｉａ：再生状態が時間に依存するメディア）クラスを定義す

る。この連続メディアクラスは上述のメディアクラスの導出クラスであるから、上記２つの状態”ＷＡＩＴ”と”ＲＵＮ”を有する。

【００５５】さらに、”ＲＵＮ”の状態を

あ）通常再生（ＰＬＡＹ）

い）早送り再生（ＦＡＳＴ＿ＰＬＡＹ）

の２つの状態に分ける。このうち、”通常再生（ＰＬＡＹ）”を初期状態とする（状態番号を０として設定した方が初期状態とされる）。

【００５６】このようなＣｏｎｔｉｎｕｏｕｓＭｅｄｉａクラスのクラス定義は次のようになる。

```
class ContinuousMedia : Media {
 state:
  1.0: PLAY;
  1.1: FAST_PLAY;
 event:
  1.0:playFast: 1.0→1.1;
  1.1:playNormal: 1.1→1.0;
 }
```

【００５７】ここで、”：Ｍｅｄｉａ”は、”ＣｏｎｔｉｎｕｏｕｓＭｅｄｉａ”クラスが導出される親クラス

を示す。また、状態番号が”ｎ．ｍ（ｎ，ｍは正の整数）”となっているのは、状態”ｎ．ｍ：．．．”が、親クラスの状態”ｎ：．．．”に含まれることを意味する。すなわち、状態”１．０：ＰＬＡＹ”と”１．１：ＦＡＳＴ＿ＰＬＡＹ”は、ともに親クラス”Ｍｅｄｉａ”の状態”１：ＲＵＮ”の中に含まれる。

【００５８】ただし、この場合、状態”１．０：ＰＬＡＹ”が初期状態（状態番号が０となっているため）となる。つまり、状態”ＷＡＩＴ”のときにイベント”ｒｕｎ”を受け取ると、状態”ＰＬＡＹ”に遷移する。また、状態”ＰＬＡＹ”または”ＦＡＳＴ＿ＰＬＡＹ”のときにイベント”ｓｔｏｐ”を受け取ると、状態”ＷＡＩＴ”に遷移する。このように、子のクラス（下位のクラス）を定義する場合に、親のクラス（上位のクラス）に定義されている状態遷移を継承させることができる。

【００５９】また、以上のＣｏｎｔｉｎｕｏｕｓＭｅｄｉａクラスの状態遷移を図と表に表わすと、図３と表２に示すようになる。

【００６０】

【表２】

〈ＣｏｎｔｉｎｕｏｕｓＭｅｄｉａクラスの状態遷移表〉

	イベント	
状態	ｐｌａｙＦａｓｔ	ｐｌａｙＮｏｒｍａｌ
ＰＬＡＹ	ＦＡＳＴ＿ＰＬＡＹ	－
ＦＡＳＴ＿ＰＬＡＹ	－	ＰＬＡＹ

【００６１】次に、上述の各クラス（ＭｅｄｉａとＣｏｎｔｉｎｕｏｕｓＭｅｄｉａ）を用いて、所定のオブジェクトの表示状態を制御するために、例えば、下記のようなスクリプト（スクリプト名”ｓａｍｐｌｅ”）を定義する。

```
script sample {
 a = new Media(content_identifier = 1);
 b = new ContinuousMedia(content_identifier = 2);
 c = new ContinuousMedia(content_identifier = 3);
 do {run→a;}
 switch (a.getMessage(run)) {
  do {run→c;}
 }
 switch (a.isSelected()) {
  case ((b.testState(0)) && (c.testState(0))) do {run→b;}
  case ((b.testState(1.0)) && (c.testState(0))) do {playFast→b;
       run→c;}
```

11　　12

```
 case ((b.testState(0)) && (c.testState(1.0))) do {run→b;
       playFast→c;}
 }
 switch (c.getMessage(stop)) {
  case ((a.testState(1) && b.testState(0)) do {stop→a;}
 }
}
```

【００６２】即ち、この実施例においては、オブジェクトａがクラスＭｅｄｉａとして規定され、オブジェクトｂとｃがクラスＣｏｎｔｉｎｕｏｕｓＭｅｄｉａとして規定される。そして、それぞれはＩＤ（ｃｏｎｔｅｎｔ＿ｉｄｅｎｔｉｆｉｅｒ）１，２または３として識別されるようになされている。

【００６３】do {run→a;}

は、オブジェクトａにイベントｒｕｎを転送することを意味している。

```
switch (a.getMessage(run)) {
    do {run→c;}
}
```

は、ａというオブジェクトがｒｕｎというメッセージを受けたとき、ｃというオブジェクトにｒｕｎというイベントを転送することを意味している。

【００６４】

```
switch (a.isSelected()) {
    case ((b.testState(0)) && (c.testState(0))) do {run→b;}
    case ((b.testState(1.0)) && (c.testState(0))) do {playFast→b;
         run→c;}
    case ((b.testState(0)) && (c.testState(1.0))) do {run→b;
         playFast→c;}
}
```

は、オブジェクトａが選択されたとき、各ｃａｓｅの条件が充足されるか否かを順次判定するものである。このうち、例えば

```
case ((b.testState(0)) && (c.testState(0))) do {run→b;}
```

は、オブジェクトｂの状態が０（ＷＡＩＴ）であり、かつ、オブジェクトｃの状態も０（ＷＡＩＴ）であるとき、オブジェクトｂに対して、ｒｕｎのイベントを転送することを意味している。

【００６５】図４に、上述のクラス定義とスクリプトを交換するマルチメディア情報表示／再生装置（情報伝送装置または情報出力装置）の構成例を示す。この実施例においては、図中左側が、クラス定義とスクリプトの作成を行う送り側３１とされ、図中右側が、表示制御を行なう受け側３２とされている。

【００６６】上述したクラス定義部のファイルとスクリプトのファイルが、送り側３１のクラス定義／スクリプト作成部１１において作成される。さらに、スクリプトにおけるｃｏｎｔｅｎｔ＿ｉｄｅｎｔｉｆｉｅｒ１，２，３に対応する表現メディアのオブジェクトが、メディアオブジェクトデータ作成部１２において作成される。例えば、ｃｏｎｔｅｎｔ＿ｉｄｅｎｔｉｆｉｅｒ１のデータとして、符号化方式Ａで符号化された静止画のデータが含まれているファイルが作成され、ｃｏｎｔｅｎｔ＿ｉｄｅｎｔｉｆｉｅｒ２のデータとして、符号化方式Ｂで符号化された音声のデータが含まれているファイルが作成され、さらに、ｃｏｎｔｅｎｔ＿ｉｄｅｎｔｉｆｉｅｒ３のデータとして、符号化方式Ｃで符号化された動画のデータが含まれているファイルが作成される。

【００６７】この例においては、スクリプトから判るように、ｃｏｎｔｅｎｔ＿ｉｄｅｎｔｉｆｉｅｒ１に対応するオブジェクトはＭｅｄｉａクラスのオブジェクトであり、ｃｏｎｔｅｎｔ＿ｉｄｅｎｔｉｆｉｅｒ２，３に対応するオブジェクトはＣｏｎｔｉｎｕｏｕｓＭｅｄｉａクラスのオブジェクトであり、符号化方式にそれぞれＡ，Ｂ，Ｃを選んでいるが、他の符号化方式を用いることができるのは勿論である。すなわち、スクリプトそのものは、表現メディアの符号化方式に依存しないため、受け側３２の能力に合わせた符号化方式を選択することができる。

【００６８】それぞれの表現メディアオブジェクトファイルのデータ構造には、ｃｏｎｔｅｎｔ＿ｉｄｅｎｔｉｆｉｅｒを記述するレコード、符号化方式を記述するレコード、表示属性を記述するレコード、および符号化されたデータが格納されるレコードがある。

【００６９】クラス定義スクリプト作成部１１により作成したクラス定義部とスクリプトのファイル、並びに、メディアオブジェクトデータ作成部１２により作成されたメディアオブジェクトのファイルは、転送制御部１３に入力され、合成される。転送制御部１３より出力されたデータは、通信制御装置１４において所定の方式で変調され、伝送媒体１０に伝送される。

13

【００７０】伝送媒体１０と転送制御プロトコルには、例えば、ＥｔｈｅｒｎｅｔとＴＣＰ／ＩＰを用いることができる。勿論、その他の組合せを用いることも可能である。

【００７１】受け側３２においては、伝送媒体１０を介して伝送されてきたデータを通信制御装置１５が受信、復調し、これを転送制御部１６に供給する。転送制御部１６は、受け取ったファイルを分離し、クラス定義部とスクリプトのファイルをスクリプト解釈／実行部１７に供給し、残りの表現メディアオブジェクトのファイルを表示制御部１８に供給する。

【００７２】スクリプト解釈／実行部１７においてクラス定義またはスクリプトが解釈される以前に、送り側３１と受け側３２の間で（具体的には送り側３１のクラス定義／スクリプト作成部１１と、受け側３２のスクリプト解釈／実行部１７の間で）、各クラスの状態の表示上の意味について合意がなされている必要がある。例えば、この実施例では、Ｍｅｄｉａクラスの状態ＷＡＩＴとＲＵＮや、ＣｏｕｔｉｎｕｏｕｓＭｅｄｉａクラスの状態ＰＬＡＹとＦＡＳＴ＿ＰＬＡＹの違いについて合意がなされている必要がある。

【００７３】本実施例においては、状態を識別したり、状態間の遷移を定義することは可能であるが、個々の状態の表示上の意味については定義されていない。従って、スクリプトの交換（伝送）に先立ち、予め各クラスの状態の意味について合意を行なうという過程が必要となるのである。

【００７４】以上のような合意が予めなされている状態で、最初にスクリプト解釈／実行部１７において、クラス定義部の解釈が行なわれ、次にスクリプトの＜オブジェクト定義部＞が解釈され、オブジェクトａ，ｂ，ｃの状態情報を記憶する領域が確保される。状態情報を記憶する領域には、オブジェクトａ，ｂ，ｃの各々の状態番号が記憶される。クラス定義から、オブジェクトａは１桁の状態番号を持ち、オブジェクトｂ，ｃは２桁の状態番号を持つことが判るので、表３に示すように、各々のオブジェクトに対して２つの整数のフィールド１，２を持つレコードが定義される。

【００７５】

【表３】

〈オブジェクトの状態記憶領域〉

オブジェクト	a		b		c	
フィールド	1	2	1	2	1	2

【００７６】スクリプト解釈／実行部１７では、すべてのオブジェクトを初期状態にセットする。すなわちオブジェクトａ，ｂ，ｃのいずれも状態”ＷＡＩＴ”になり、すべてのフィールドは”０”で初期化される（表４の状態ｎｏ．１参照）。

14

【００７７】

【表４】

〈オブジェクトの状態遷移の例1〉

オブジェクト	a		b		c	
フィールド	1	2	1	2	1	2
状態ｎｏ．１	0	0	0	0	0	0
状態ｎｏ．２	1	0	0	0	0	0
状態ｎｏ．３	1	0	0	0	1	0
状態ｎｏ．４	1	0	1	0	1	1
状態ｎｏ．５	1	0	0	0	0	0
状態ｎｏ．６	0	0	0	0	0	0

【００７８】このとき、スクリプト解釈／実行部１７において、スクリプトの＜イニシャルメッセージパッシング記述部＞が解釈され、”ｒｕｎ”という名前のイベントがオブジェクトａに送られる。

【００７９】オブジェクトａは、上述したように、次のような文により、Ｍｅｄｉａクラスのオブジェクトと宣言されている。

```
a = new Media(content_identifier = 1);
```

このため、Ｍｅｄｉａクラスの定義により、イベント”ｒｕｎ”を受け取ると、状態が０から１、すなわち状態”ＷＡＩＴ”から状態”ＲＵＮ”に遷移することになる。従って、オブジェクトａの状態情報を記憶する領域の第１フィールドには状態番号”１”が記憶される（表４における状態ｎｏ．２参照）。

【００８０】このときまた、スクリプト解釈／実行部１７は、表示制御部１８に対して、オブジェクトａの内容部を表示するよう要求する。これにより、表示制御部１８は、ｃｏｎｔｅｎｔ＿ｉｄｅｎｔｉｆｉｅｒ＝１に対応する表現メディアオブジェクトのデータ（符号化方式Ａにて符号化されている）を、内蔵する復号器（符号化

15

方式Ａに対応する、例えばＭＲ（ｍｏｄｉｆｉｅｄ　ｒｅａｄ）復号器）により復号化し、内蔵するディスプレイに表示させる。

【００８１】このように、あるイベントがあるオブジェクトに送られ、その対象オブジェクトの状態遷移が起こる事象を、”イベントの転送が検出される”という。スクリプト解釈／実行部１７は、表示制御部１８によりイベント転送が検出された場合、スクリプトの＜イベントメッセージパッシング記述部＞の＜イベント判定部＞を１つずつテストする。

【００８２】いまの場合、最初の文

```
switch (a.getMessage(run)) {
  do {run→c;}
}
```

の＜イベント判定部＞に該当するイベントが起こったことになるため、ｄｏ以降の＜メッセージパッシング記述部＞

```
do {run→c;}
```

が実行される。すなわち、”ｒｕｎ”という名前のイベントがオブジェクトｃに送られる。

【００８３】オブジェクトｃは文

```
c = new ContinuousMedia(content_identifier = 3);
```

により、ＣｏｎｔｉｎｕｏｕｓＭｅｄｉａクラスのオブジェクトと宣言されているが、ＣｏｎｔｉｎｕｏｕｓＭｅｄｉａクラスの定義には、イベント”ｒｕｎ”を受け取った場合の状態遷移の定義がなされていない。しかしながら、このＣｏｎｔｉｎｕｏｕｓＭｅｄｉａクラスの親クラスがＭｅｄｉａクラスであるので、Ｍｅｄｉａクラスに定義されているイベント”ｒｕｎ”を受け取った場合の状態遷移定義に従い、オブジェクトｃの状態が状態”ＲＵＮ”、すなわち状態”ＰＬＡＹ”になる。なぜなら、ＣｏｎｔｉｎｕｏｕｓＭｅｄｉａクラスの定義に

```
case ((b.testState(0)) && (c.testState(1.0))) do {playFast→c;}
```

に該当するので（オブジェクトｂが状態”ＷＡＩＴ”であり、かつ、オブジェクトｃが状態”ＰＬＡＹ”であるので）、

＜メッセージパッシング記述部＞

```
do {playFast→c;}
```

が実行され、”ｐｌａｙＦａｓｔ”という名前のイベントがオブジェクトｃに送られる。

【００８８】これにより、オブジェクトｃは状態が、”ＰＬＡＹ”から”ＦＡＳＴ＿ＰＬＡＹ”に遷移する。従って、オブジェクトｃの状態情報を記憶する領域に、状態番号”１．１”が記憶される（表４の状態ｎｏ．４参照）。またこのとき、オブジェクトｃの動画の再生速度を早くする指令が、スクリプト解釈／実行部１７から表示制御部１８に通知され、表示制御部１８において、オブジェクトｃの動画が高速再生される。

【００８９】この後、オブジェクトｃの再生が終了する（データの最後まで再生してしまう）と、オブジェクト

16

より、親クラスの状態”ＲＵＮ”の初期状態が状態”ＰＬＡＹ”になると宣言されているためである。

【００８４】そして、オブジェクトｃの状態情報を記憶する領域の第１フィールドに状態番号１が、第２フィールドに状態番号０が記憶される。すなわち、状態番号”１．０”が記憶されることになる（表４の状態ｎｏ．３参照）。このとき、スクリプト解釈／実行部１７は、表示制御部１８に対して、オブジェクトｃの内容部、すなわち、ｃｏｎｔｅｎｔ＿ｉｄｅｎｔｉｆｉｅｒ＝３に対応する表現メディアオブジェクトを表示するよう要求する。表示制御部１８は、この要求に対応して、符号化方式Ｃの復号器により動画データを復号し、ディスプレイに表示させる。

【００８５】ここで、オブジェクトｃの動画の内容が再生されている途中で、ユーザがディスプレイ上で、例えばマウス等の表示制御部１８の管理下の指示装置により、オブジェクトａを指示したとすると、オブジェクトスクリプト解釈／実行部１７にイベント転送の検出が通知（この場合のイベントは表示状態の遷移を起こさない特別なものである。つまり、オブジェクトａの表示領域内においてユーザの指示が起こったという事象のみが通知）され、スクリプトの＜イベントメッセージパッシング記述部＞の＜イベント判定部＞がテストされる。

【００８６】このとき、このイベントが、

```
switch (a.isSelected()) {
}
```

に該当するイベントであることが判るため、このｓｗｉｔｃｈ文の中の各ｃａｓｅ文の＜状態判定部＞がテストされる。

【００８７】いまの場合、

ｃに対して”ｓｔｏｐ”というイベントが送られ、オブジェクトｃの状態が状態”ＷＡＩＴ”に遷移し、状態番号が”０”に設定される（表４の状態ｎｏ．５参照）。

【００９０】この”ｓｔｏｐ”イベントの転送は、スクリプトそのものには記述されていない特別なもので、時間に依存する連続メディアの再生が終了したときに限り、表示制御部１８が”ｓｔｏｐ”イベントの転送が起こったものとみなし、スクリプト解釈／実行部１７に対してイベント転送の検出を通知する。

【００９１】このとき、スクリプトの＜イベントメッセージパッシング記述部＞の＜イベント判定部＞がテストされ、このイベントが、

```
switch (c.getMessage(stop)) {
}
```

に該当するイベントであることが判るため、このｓｗｉｔｃｈ文の中の各ｃａｓｅ文の＜状態判定部＞がテストされる。

【００９２】いまの場合、オブジェクトａが状態”ＰＬＡＹ”であり、かつ、オブジェクトｂが状態”ＷＡＩＴ”であるから、

```
case ((a.testState(1) && b.testState(0)) do {stop→a;}
```

の＜メッセージパッシング記述部＞

```
do {stop→a;}
```

が実行され、オブジェクトａの表示がディスプレイから消えて、状態が状態”ＷＡＩＴ”に遷移する（表４の状態ｎｏ．６参照）。

【００９３】上述の表４の状態ｎｏ．３において、オブジェクトｃの再生が終了してしまった後に、ユーザがディスプレイ上でオブジェクトａを指示した場合の表示の流れは、表５に示すようになる。

【００９４】

【表５】

〈オブジェクトの状態遷移の例２〉

オブジェクト	a		b		c	
フィールド	1	2	1	2	1	2
状態ｎｏ．３	1	0	0	0	1	0
状態ｎｏ．４	1	0	0	0	0	0
状態ｎｏ．５	0	0	0	0	0	0

【００９５】オブジェクトｃの再生が終了した場合、オブジェクトｃに対して”ｓｔｏｐ”というイベントが送られたものとみなされ、オブジェクトｃの状態が”ＷＡＩＴ”に遷移し、状態番号が”０”に設定される（表５の状態ｎｏ．４参照）。スクリプト解釈／実行部１７にイベント転送の検出が通知され、オブジェクトａの内容がディスプレイから消えて、状態が”ＷＡＩＴ”に遷移する（表５の状態ｎｏ．５参照）。

【００９６】従って、以後は、ユーザがディスプレイ上でマウス等の指示装置により、オブジェクトａを指示することができないため、オブジェクトの状態遷移の表４の場合と異なる状態の流れとなる。

【００９７】以上の処理をフローチャートに表すと、図５および図６に示すようになる。即ち、図５に示すように、最初にステップＳ１において、クラス定義部の解釈を行い、次にステップＳ２において、スクリプトの解釈／実行を行う。

【００９８】このステップＳ２におけるスクリプトの解釈／実行の処理の詳細は、図６に示されている。

【００９９】即ち、最初にステップＳ１１において、オブジェクト定義部の解釈が実行される。次にステップＳ１２に進み、ステップＳ１１におけるオブジェクト定義部の解釈に対応して、オブジェクト状態領域の確保が実行される。さらにステップＳ１３においては、ステップＳ１２において確保したオブジェクト状態記憶領域が初期化される。

【０１００】次にステップＳ１４において、イニシャルメッセージパッシング記述部が実行され、そしてステップＳ１５において、オブジェクト状態が更新される。

【０１０１】ステップＳ１６においては、ｅｖｅｎｔ＿ｃｈｅｃｋ＿ｃｏｕｎｔｅｒが１にセットされる。そして、次にステップＳ１７に進み、ｅｖｅｎｔ＿ｃｈｅｃｋ＿ｃｏｕｎｔｅｒ番目（いまの場合、この値は、ステップＳ１６において１に設定されている）のイベント判定部のテストが実行される。テストの結果が正しいとき、ステップＳ１８に進み、ｓｔａｔｅ＿ｃｈｅｃｋ＿ｃｏｕｎｔｅｒに１が設定される。

【０１０２】次にステップＳ１９に進み、ｓｔａｔｅ＿ｃｈｅｃｋ＿ｃｏｕｎｔｅｒ番目（いまの場合、この値は、ステップＳ１８において１に設定されている）の状態判定部のテストが実行される。このテストの結果が正しいとき、ステップＳ２０に進み、ｓｔａｔｅ＿ｃｈｅｃｋ＿ｃｏｕｎｔｅｒ番目のメッセージパッシング記述部が実行される。

【０１０３】ステップＳ２０における処理の後、ステップＳ２１に進み、ｓｔａｔｅ＿ｃｈｅｃｋ＿ｃｏｕｎｔｅｒの値が１だけインクリメントされる。ステップＳ１９において、ｓｔａｔｅ＿ｃｈｅｃｋ＿ｃｏｕｎｔｅｒ番目の状態判定部のテストの結果が誤りであると判定された場合においては、ステップＳ２０の処理はスキップされ（そのメッセージパッシング記述部は実行されず）、ステップＳ２１の処理が実行される。

【０１０４】次にステップＳ２２に進み、ｓｔａｔｅ＿ｃｈｅｃｋ＿ｃｏｕｎｔｅｒの値が１つのｓｗｉｔｃｈ文における状態判定部の総数と等しいか否かが判定される。ステップＳ２２における判定がＮＯである場合においては、ステップＳ１９に戻り、それ以降の処理が繰り返される。ステップＳ２２における判定がＹＥＳである場合においては、ステップＳ２３に進み、ｅｖｅｎｔ＿ｃｈｅｃｋ＿ｃｏｕｎｔｅｒを１だけインクリメントする。ステップＳ１７において、ｅｖｅｎｔ＿ｃｈｅｃｋ＿ｃｏｕｎｔｅｒ番目のイベント判定部のテストの結果が誤りであると判定された場合においては、ステップＳ１８乃至Ｓ２２における処理がスキップされ、ステップＳ２３の処理が実行される。

【０１０５】ステップＳ２３の次にステップＳ２４に進

み、ステップＳ２３でインクリメントしたｅｖｅｎｔ＿ｃｈｅｃｋ＿ｃｏｕｎｔｅｒの値がイベント判定部の総数と等しいか否かが判定される。ステップＳ２４の判定がＮＯである場合においては、ステップＳ１７に戻り、それ以降の処理が繰り返し実行される。ステップＳ２４の判定がＹＥＳである場合においては、ステップＳ２５に進み、イベント転送の検出待ちの状態となる。そして、イベント転送が検出されたとき、ステップＳ２５からステップＳ１５に戻り、それ以降の処理が繰り返し実行される。

【０１０６】以上においては、送り側３１から受け側３２に対して、伝送媒体１０を介してクラス定義とスクリプトを伝送するようにしたが、例えば図７に示すように、蓄積媒体２０を介して送り側３１から受け側３２に対して、これらのデータを伝送することが可能である。この場合においては、図４の実施例における通信制御装置１４，１５が、蓄積媒体制御装置２１，２２に変更される。蓄積媒体制御装置２１は、転送制御部１３より供給されるデータを蓄積媒体２０に記録し、蓄積媒体制御装置２２は、蓄積媒体２０に記録されたデータを再生し、転送制御部１６に出力する。

【０１０７】以上においては、クラス定義／スクリプトの交換に先立ち、予め各クラスの状態の表示上の意味について、送り側と受け側の間で、合意がなされていなければならないことを述べたが、これは、例えば、標準化機関等において、クラス定義の登録手続きを規定し、クラス名により一意に参照可能なクラス定義と、そこで記述されている状態の表示上の意味を記述したドキュメントを登録することにより実現することができる。

【０１０８】このようにすれば、クラス定義の部分を広範囲の異なるシステムにおいて共有することができる。また、スクリプトの汎用性（さまざまなベンダーからのシステム／プラットフォーム上でのスクリプトの互換性）が保証されることとなる。

【０１０９】

【発明の効果】以上の如く、本発明の情報同期制御方法、情報伝送装置または情報出力装置によれば、各情報のメディアからの出力状態と、出力状態の遷移を階層的に定義し、出力状態とその遷移を利用したスクリプトをもとに、各情報の出力状態の同期を制御するようにしたので、マルチメディアオブジェクトのより細かな同期制御が可能となる。

【図面の簡単な説明】

【図１】クラスの階層的定義による状態遷移の承継を説明する図である。

【図２】メディアクラスの状態遷移を説明する図である。

【図３】連続メディアクラスの状態遷移を説明する図である。

【図４】本発明の情報伝送装置および情報出力装置の一実施例の構成を示すブロック図である。

【図５】図４の実施例の動作を説明するフローチャートである。

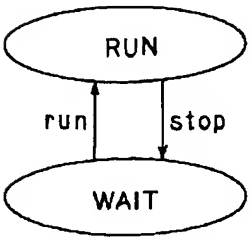
【図６】図５のステップＳ２のより詳細な処理を説明するフローチャートである。

【図７】本発明の情報伝送装置および情報出力装置の他の実施例の構成を示すブロック図である。

【符号の説明】

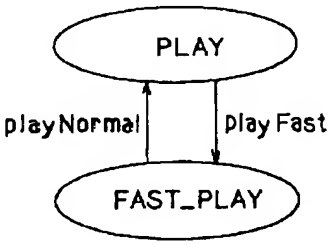
１１　クラス定義スクリプト作成部
１２　メディアオブジェクトデータ作成部
１３　転送制御部
１４，１５　通信制御装置
１６　転送制御部
１７　スクリプト解釈／実行部
１８　表示制御部
２０　蓄積媒体
２１，２２　蓄積媒体制御装置

【図２】



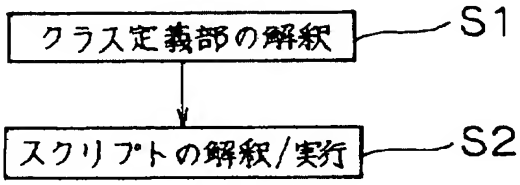
メディアクラスの状態遷移図

【図３】



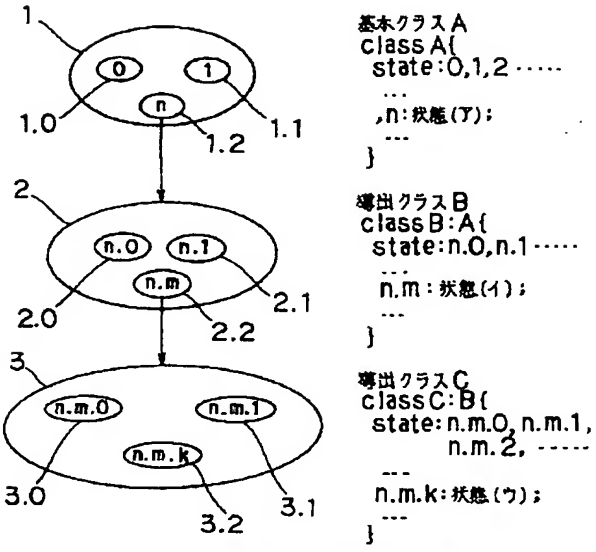
連続メディアクラスの状態遷移図

【図５】



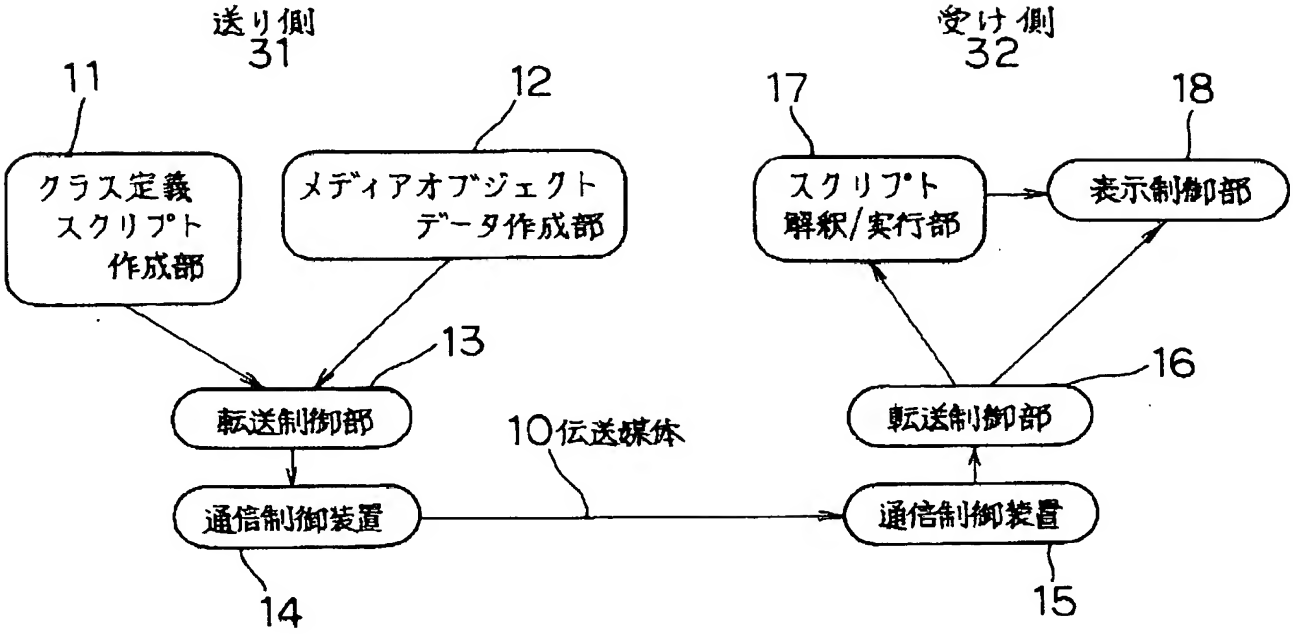
処理のアルゴリズム

【図１】



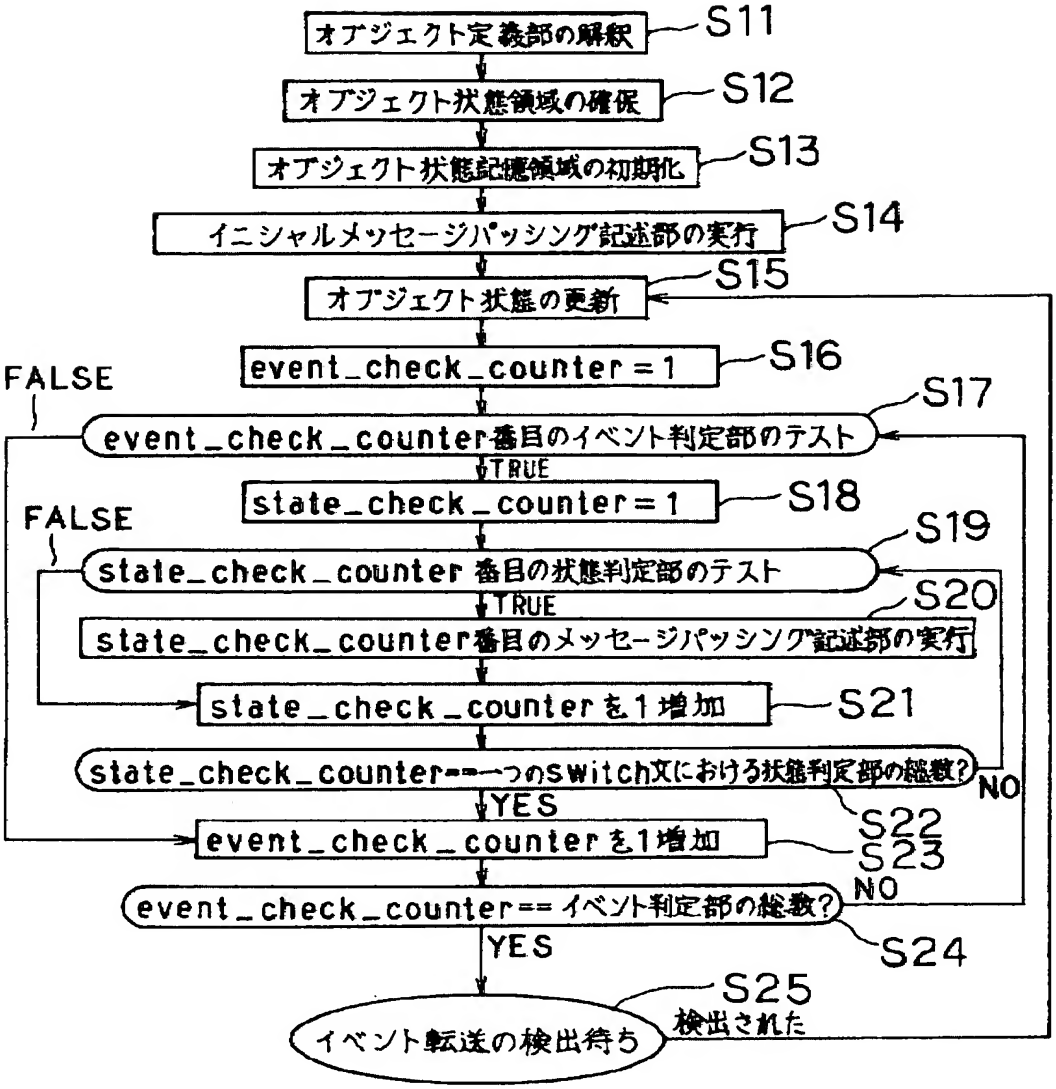
クラスの階層的定義による状態遷移の継承

【図４】



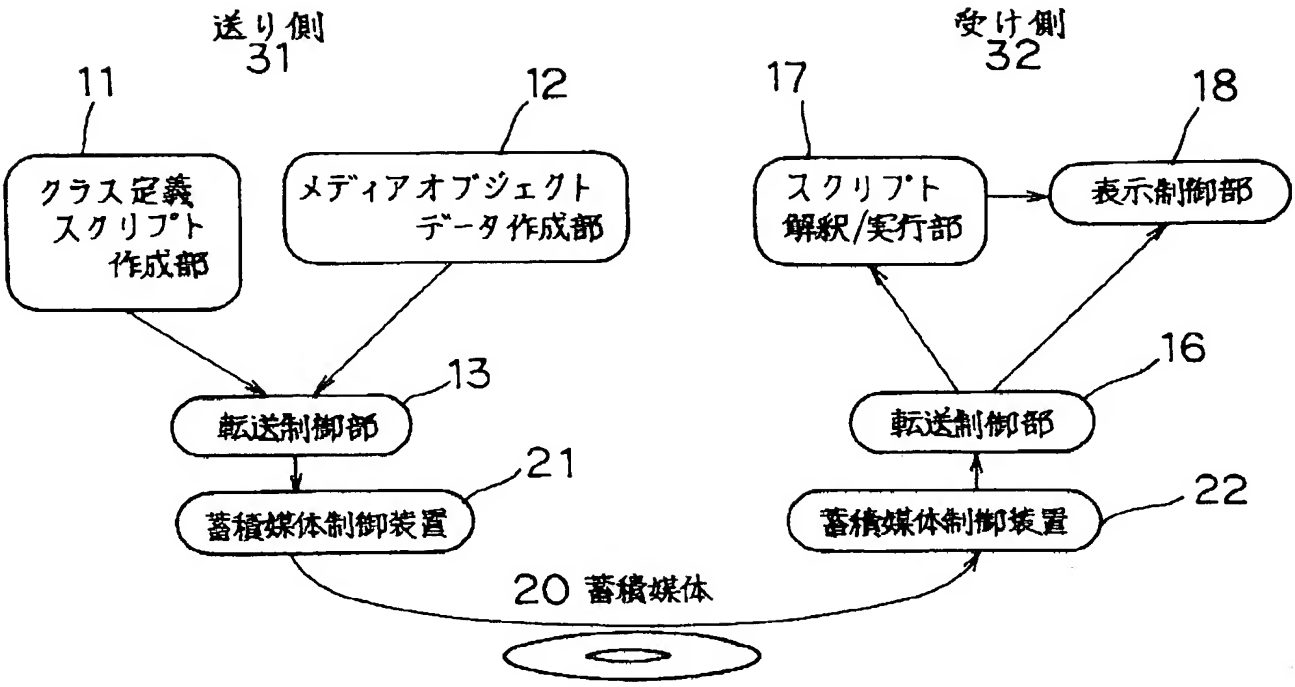
具体例１のシステム構成

【図６】



スクリプトの解釈/実行

【図７】



具体例２のシステム構成